Remarks

As stated above, Applicants appreciate the Examiner's thorough examination of the subject application and request reexamination and reconsideration of the subject application in view of the preceding amendments and the following remarks.

As of the office action of June 13, 2008, claims 1-31 were pending in the subject application. With this response applicant has amended claim 11.

A. 35 U.S.C. § 112 Rejection

The Examiner has rejected claim 11 under 35 U.S.C. § 112 as indefinite. In particular, the Examiner states that "[o]ne of ordinary skill in the art would not understand what is meant by [the term] 'intrinsic[.]'" Office Action page 2.

In response, Applicants have amended claim 11 to remove the term "intrinsic." The newly amended claim is listed below for convenience:

> 11. (Currently Amended) The method of claim 1, wherein the allotted playback duration is determined based upon predetermined rights associated with the device.

As noted, the term "intrinsic" has been removed. The new claim recites, in part, "predetermined rights associated with the device." The amendment is supported by the specification. For example, paragraph [0020] states that "digital playback devices *equipped with intrinsic digital content consumption rights* are provisioned with rights monitoring logic to influence playback" Subject Application ¶ [0020] (emphasis added). As another example, paragraph [0024] states:

> [t]he right representing the allotted playback duration may be intrinsic to the playback device, or it may be represented by rights provided to the playback device by the content provider or third party as part of a subscription or other transactional agreement.

Id. at ¶ [0024]. Applicants believe that the newly amended claim, in view of the specification, is patentable under § 112. Accordingly, Applicants request withdrawal of the § 112 rejection of claim 11.

B. 35 U.S.C. § 102 Rejections

The examiner has rejected claims 1-5 and 10-31 under 35 U.S.C. § 102(e) over U.S. Patent Application No. 2005/0022019 (filed July 5, 2003) ("Medvinsky"). Applicants do not believe that Medvinsky claims each and every element claimed in the subject application. Additionally, Applicants respectfully assert that Medvinsky is not prior art under § 102(e) because the invention of the subject application pre-dates the Medvinsky's reference. In support of this assertion, Applicants have attached a 37 C.F.R. § 1.131 Affidavit of inventor Joshua Hug as Appendix A (the "Hug Affidavit"), and a 37 C.F.R. § 1.131 Affidavit of inventor Bradley Hefta-Gaub as Appendix B (the "Hefta-Gaub Affidavit") (collectively, "the affidavits").

The affidavits provide evidence that the invention pre-dates the Medvinsky reference. In particular, the affidavits provide evidence that conception occurred prior to the Medvinsky filing date, and that diligent reduction to practice began prior to the Medvinsky filing date and continued, without lapse, through the subject application's filing date. The attached affidavits state that "[t]he subject matter claimed in the subject application was conceived prior to July 5, 2003." Hefta-Gaub Affidavit, ¶ 4; Hug Affidavit ¶ 4. The affidavits also include copies of an Invention Disclosure Statement that describes the claimed invention and pre-dates the Medvinsky reference. *See Id.* at ¶ 4-5.

In addition, the inventors diligently pursued reduction to practice of the claimed invention. As stated in the affidavits, the reduction to practice began prior to the Medvinsky filing date and continued, without lapse, though the filing date of the subject application. *See Id.*

at ¶ 6. The affidavits include emails and attachments that evidence the reduction to practice—the emails are dated July 1, 2003 through October 8, 2003. *See Id.* at ¶ 6-13. These emails, coupled with the filing of the subject application, attest to the diligent reduction to practice of the invention which began before the Medvinsky filing and continued, without lapse, through the filing date of the subject application.

The Examiner will note that Exhibit E, attached to this response and referenced in the affidavits, contains an email message and two PowerPoint® file attachments ("PPT files"). Each page of the PPT files displays the date September 30, 2008. That date is the print-out date of the PPT files; it is the day that Applicants' attorney printed the files—it is not the date that the files were created. As noted in the affidavits, the PPT files were originally attached to the email within Exhibit E, which is dated August 15, 2003.

Accordingly, Applicants contend that the Medvinsky reference is not prior art under § 102(e) because, as discussed, the invention of the subject application pre-dates the filing of the Medvinsky reference. As such, Applicants believe that Medvinsky cannot provide proper basis for the § 102(e) rejections. In light of the evidence presented, Applicants respectfully request withdrawal of the § 102(e) rejections.

C. 35 U.S.C. § 103 Rejections

The Examiner has rejected claims 6-8 under 35 U.S.C. § 103 over Medvinsky and U.S. Patent No. 5,586,264 ("Belknap"), and claim 9 under § 103 over Medvinsky and U.S Patent No. 5,708,422 ("Blonder"). Claims 6-9 are ultimately dependent upon independent claim 1, which was rejected under § 102 over Medvinsky. As discussed above, Applicants contend that claim 1 is patentable under § 102 because the invention pre-dates the Medvinsky reference. Applicants further contend that claims 6-9 are patentable because they are ultimately dependent upon claim

1 and incorporate all the elements of claim 1. Accordingly, Applicants respectfully request withdrawal of the § 103 rejection of claims 6-9.

D. Conclusion

In consideration of the amendments and foregoing discussion, the application is now believed to be in condition for allowance. Early allowance of the subject application is respectfully solicited.

This response is not believed to necessitate any additional fees. However, in the event that additional fees are due, please charge or credit any refund to our Deposit Account No. 50-2324.

Respectfully Submitted,

Dated: 13 November 2008

/Brian J Colandreo/
Brian J Colandreo
Reg. No. 42,427

Holland & Knight LLP
10 St. James Avenue
Boston, MA 02116-3889
Telephone 617-305-2143
Facsimile 617-523-6850
5670429_v1

EXHIBIT A

REALNETWORKS INVENTION DISCLOSURE FORM

ATTORNEY-CLIENT PRIVILEGED COMMUNICATION

DATE: [redacted]

It is important to provide accurate and detailed information on this form. The information will be used to evaluate your invention for possible filing as a patent application and is necessary to help us complete the application. When completed and signed, please return this form to Lisa Benner, **Legal Department**. If you have any questions, please call Steven Stewart, x6467.

1. Inventor: Hug (Last Name) Joshua (First Name) D (Middle Initial)

Phone 206-892-6621 Location: Seattle Fax #

Citizenship: USA Contractor: YES NO X

Inventor E-Mail Address: jhug@real.com

Home Address: 1101 Queen Anne Ave. APT 202

City Seattle State WA Zip 981009 Country USA

*Corporate Level Division (e.g. Media Systems) Media Systems Sub -Division Client Dev Group

Supervisor* Rahul Agarwal Phone 206-674- 2359 Location: Seattle

Inventor: Hefta-Gaub (Last Name) Bradley (First Name) D (Middle Initial)

Phone 206-674-2272 Location: Seattle Fax #

Citizenship: USA Contractor: YES NO X

Inventor E-Mail Address: brad@real.com

Home Address: 3939 Eastern Ave. N.

City Seattle State WA Zip 98103 Country USA

*Corporate Level Division (e.g. Media Systems) Media Systems Sub -Division Codec

Supervisor* Martin Plaehn Phone Location: Seattle

***If you are unsure of this information, please discuss with your manager.**

(PROVIDE SAME INFORMATION AS ABOVE FOR EACH ADDITIONAL INVENTOR)

2. Title of Invention: Method of expiring and renewing subscription media

3. Describe the technology/product/process (code name) the invention relates to (be specific if you can): Portable music & media subscription devices.

4. Include several key words to describe the technology area of the invention in addition to # 3 above: DRM, Rights enforcement, secure transfer, portable devices, rights provisioning

5. Stage of development (i.e. % complete, simulations done, alpha code if any, etc.): Idea has been documented in slides, no code has been created nor has this been shared with an external party.

6. (a) Has a description of your invention been, or will it shortly be, published outside of Real Networks?

NO: X YES: If YES, was the manuscript submitted to Legal for pre-publication approval?

IDENTIFY THE PUBLICATION AND THE DATE PUBLISHED (TO BE PUBLISHED):

(b) Has your invention been used/sold or planned to be used/sold by Real Networks or others?

NO: X YES: DATE WAS OR WILL BE SOLD:

(c) Does this invention relate to technology that is or will be covered by a SIG (special interest group)/standard/ or specification?

NO: _____ YES: X Name of SIG/Standard/Specification: Future Open mobile alliance DRM standards... (not currently in scope, but after we demonstrate I assume they will jump all over this.)

(d) If the invention is software, actual or anticipated date of any beta tests outside RealNetworks: Late Summer/Fall 2003

7. Was the invention conceived or constructed in collaboration with anyone other than a RealNetworks employee or in performance of a project involving entities other than RealNetworks, e.g. government, other companies, universities or consortia?

NO: X YES: _____ Name of individual or entity: _____

8. Is this invention related to any other invention disclosure that you or anyone else has recently submitted? If so, please give the title and inventors names: No.

PLEASE READ AND FOLLOW THE DIRECTIONS ON HOW TO WRITE A DESCRIPTION OF YOUR INVENTION

Attach a description of the invention to this form, DATED AND SIGNED BY AT LEAST ONE PERSON WHO IS NOT A NAMED INVENTOR, and include the following information:

1. **Describe in detail what the components of the invention are and how the invention works.**
2. **Describe advantage(s) of your invention over what is done now. (E.g. What problems does it solve, if any?)**
3. **YOU MUST include at least one figure illustrating the invention. If the invention relates to software, include a flowchart or pseudo-code representation of the algorithm.**
4. **Value of your invention to RealNetworks (how will or could it be used?).**
5. **Explain how your invention is novel. If the technology itself is not new, explain what makes it different.**
6. **Identify the closest or most pertinent prior art that you are aware of from RealNetworks or anyone else.**
7. **Who is likely to want to use this invention or infringe the patent if one is obtained and how would infringement be detected?**

***HAVE YOUR SUPERVISOR READ, DATE AND SIGN COMPLETED FORM**

DATE: _____ SUPERVISOR: _____

BY THIS SIGNING, I (SUPERVISOR) ACKNOWLEDGE THAT I HAVE READ AND UNDERSTAND THIS DISCLOSURE

1. **Describe in detail what the components of the invention are and how the invention works.**

This invention applies the concept of a rechargeable battery to Digital Media Expiration. Expiring content in an acceptable manor is a very hard problem. This invention creates a simple yet highly effective solution to this problem.

There are two primary actors involved in this invention, (a) device (b) digital media service (DMS). An optional actor (c) is a pass through intermediary device (ie a Personal computer).

Linking the actors is a communication protocol. These Actors have been linked together in prior art. This invention focuses on the goal of the communication protocol these actors communicate through. Prior art has tried to transfer actual rights information from a DMS to a device. This has proven, (a) hard to define in a secure manor, (b) hard to create devices with the required components (c) expensive (d) a complicated user experience.

This invention removes the need of a DMS to provision rights to a device. Instead, a simple recharge command is all that needs to be transmitted. Device's will be manufactured/upgraded to contain an intrinsic set of rights they know how to enforce. Those rights will never change, and will always be enforced when a device consumes content.

Our invention is to create devices with set rights already in the devices. When these rights are used up, the device will require users to recharge their rights. The DMS will simply need to generate a refresh token for the device they want to recharge.

The most common and maybe the only set of rights that a device will acknowledge will be playback time. Again this is related to the idea of using a battery for digital rights expiration. Devices will be created with a pre-determined number of allowed playback hours. After that time has passed, the devices will refuse to play content until they are re-charged. Recharging a device can be a very simple straightforward process for all actors involved.

The following flow is imagined:

- Obtain a new device
- Register device with a DMS
- DMS sends a recharge token to the device
    - Numerous protocols are possible some of which are claims of this invention
    - Any communication medium should be usable: WiFi, WAN, GPRS, Lan, USB, 1394, SMS, MMS etc.
- Freely transfer subscription media to devices.
    - No protocol is required
    - Again any communication medium, examples are: WiFi, WAN, GPRS, Lan, USB, 1394, etc.
- Play the media consuming the intrinsic rights the device has

- At some point either automatically or at the users request, the device will be issued a refresh token.
- If the user refuses to refresh/recharge their device, does not connect their device to a DMS, cancels their subscription to their DMS, hard re-sets their device, etc, The device will refuse to playback the user's subscription content. Users will understand this because the device will behave as if the battery has run out on their device.

Protocol claims:
Numerous methods can be imagined to transfer a refresh command to a device. The protocol must ensure it cannot be re-played by a man in the middle. The DMS must be able to authenticate the identity of the device. The device must authenticate a valid DMS is re-charging/refreshing their rights.
This could be accomplished via:

- A shared secrete between the DMS & device
- A bi-directional public/private key exchange between the DMS & device
- A one-way public-key/certificate exchange between the DMS and the device.
- Or a one-way public-key/certificate exchange between the device and DMS.
- A symmetric key exchange protocol between the DMS and the device.
- The protocols above but replacing the DMS with an intermediary device (ie a PC).
- Any other protocol that full-fills the requirements listed above.

2. Describe advantage(s) of your invention over what is done now. (E.g. What problems does it solve, if any?)

This invention has numerous advantages over what is done now.

This invention greatly reduces the engineering requirements and BOM required to create secure subscription devices. Thus in the end lowering the cost users will have to pay for these devices and increasing the profit margin for manufactures of subscription devices. Currently the industry is focused on building secure clocks into secure devices. Some companies have been pushing this for a LONG time but the engineering problems have been significant. Solutions have been proposed, but they have been slow to become accepted by device manufactures because they have required devices to be manufactured with secure clocks. Secure clocks have been VERY slow to become accepted because of their impact to the BOM.

This invention simplifies life greatly for users. This is the core advantage. After the production and engineering problems are eventually solved which they are close to being solved, an even larger problem looms on the horizon. The focus has been on putting a drm on the device instead of creating a usable device. Users are not going to like or understand the idea that they have clock restrictions instead of usage restrictions on their media consumption. This invention though, uses a model that users understand, are familiar with and have already accepted.

3. YOU MUST include at least one figure illustrating the invention. If the invention relates to software, include a flowchart or pseudo-code representation of the algorithm.

The following flow is imagined:

- Obtain a new device

- Register device with a DMS
- DMS sends a recharge token to the device
    - Numerous protocols are possible some of which are claims of this invention
    - Any communication medium should be usable: WiFi, WAN, GPRS, Lan, USB, 1394, SMS, MMS etc.
- Freely transfer subscription media to devices.
    - No protocol is required
    - Again any communication medium, examples are: WiFi, WAN, GPRS, Lan, USB, 1394, etc.
- Play the media consuming the intrinsic rights the device has
- At some point either automatically or at the users request, the device will be issued a refresh token.
- If the user refuses to refresh/recharge their device, does not connect their device to a DMS, cancels their subscription to their DMS, hard re-sets their device, etc, The device will refuse to playback the user's subscription content. Users will understand this because the device will behave as if the battery has run out on their device.

4. Value of your invention to RealNetworks (how will or could it be used?).

We will use this invention to partner with device manufactures and create the subscription service of the future. We can lock others out of using similar usable technology and give our subscription services a huge competitive edge.

5. Explain how your invention is novel. If the technology itself is not new, explain what makes it different.

No rights are transferred from the DMA to the device. Instead a simple refresh message is transferred and Devices are created in a less general but more single purposed format.

6. Identify the closest or most pertinent prior art that you are aware of from RealNetworks or anyone else.

Microsoft's PD protocol. See novelty of our approach for differentiation. Prior work we have done with Portable devices has centered on the idea of transferring rights.

Sony OMG, again they rely on a clock on their devices to expire content.

One-way protocols used to bind content to a device. No expiration of content is done on devices thus these protocols are not useful for subscription services that require content to expire at some point in the future.

Who is likely to want to use this invention or infringe the patent if one is obtained and how would infringement be detected?

Microsoft, Pressplay, music match, apple. Infringement should be easily detectable.

EXHIBIT B

Subject: spec for user license request
Date: Tue, 01 Jul 2003 14:43:28 -0700
From: Qiang Luo <qluo@real.com>
To: Adam Cappio <adamc@real.com>, Sheldon Fu <sfu@real.com>, Josh Hug <jhug@real.com>, Alain Hamel <ahamel@real.com>, Brent Newman <bnewman@real.com>, Rahul Agarwal <rahul@real.com>

When making machineLicense request to the license server, the storefront shall format it as a subscription request where Subscription_SubscriptionGUID=UserGUID.

There will be one additional parameter, Subscription_ContentKey=userKey. The userKey is retrieved from the licensed user's account. The user key is encrypted with the packager's public key.

I propose that we make Subscription_Title=machineLicense mandatory for all machineLicense.

Sheldon asked me to design a new interface to get a list of all userGUIDs that are licensed on the PC. This is required in the Orange protocol. We should be able to easily implement this with GetAllRecordInfo(RT_SUBSCRIPTION) call then matching each subscription's title for "machineLicense".

Qiang

EXHIBIT C

Subject:SOD#2: User Level Licensing
Date: Fri, 11 Jul 2003 15:12:57 -0700
From: Qiang Luo <qluo@real.com>
To:drm-dev@real.com

Attached is the revised SOD for user level licensing model edited by Josh. It is based on the initial version and the follow-up discussions that I had with Josh, Sheldon and Adam.

We are having some ongoing discussions about the design of "license insertion acknowledgement" and where and when to insert the embedded content license. In fact the designs in this SOD are not finial. All in the SOD are subject to discussion, debate and change.

Qiang

SOD: User Level Licensing
11/07/2003
Version v0.2

1 Introduction:

Our current DRM implementation allows content-only licensing and subscription licensing. The licensing models have a set of rich features to support very flexible content rights control. However, both models require an individual content license for every piece of secured DRM contents. This requirement puts heavy burden on the license server in generating content licenses. It also incurs heavy client CPU load when decrypting and inserting the content licenses into the client's license database. In some rare cases, a user with a large collection of contents may somehow end up with a corrupted DB due to power failure. To recover, the user must re-license for all his contents.

To facilitate a better user experience and to address the above problems, this SOD details the functional design of a user oriented licensing model. In the user model, content is personalized to a user; the user's machine is then enabled with the user's credentials allowing consumption of the content.

Under this licensing model, the content is user-bound. A user can license up to N machines for his content. When a user is licensed on a machine that license is both user-bound and machine-bound, using the same mechanisms of subscription or normal license binding. Content bound to user A stored on the machine B will require user A's license on Machine B in order to be functional.

User licensing is an extension of our subscription licensing. A user license is simply a subscription license with a key. There are then no sublicenses to govern access to the content. Instead the content contains its content key in its file header. The user license's rights then act as a single pool of rights for the UserContent's to be accessed via. Users are no longer required to individually license individual pieces of media content.

2 Definition/Terminology

UserGUID: a 16-byte GUID assigned to a user identifying the user license in this licensing model.
UserName: the user's login name.
ClientGUID: a 16-byte DRM client permanent ID extracted from MachineInfo.
userKey: a 16-byte key assigned to the licensed user to encrypt the ContentKeys.
userLicense: the license that grants a user to play DRM contents on a machine.
embeddedContentLicense: the personalized content license in the content file header that ties the content to a particular user.

3 How the user licensing model works

When a user signs up for the service, the store account manager will assign the licensed user a UserGUID and a userKey and save them together with the normal user account information into the account DB. The userKey shall be encrypted with the packager's client public key.

When a user initiates content download, the storefront will first lookup the UserGUID and the userKey from the user's account record. The storefront will then retrieve the ContentKey using the ContentGUID in the usual way. The storefront makes a request to the license server for an embeddedContentLicense and sends both the content and the license to the client. The download manager on the client side will be responsible to insert the embeddedContentLicense into the content file header.

On playing back, the DRM client checks the content file header for the existence of the embeddedContentLicense. If present, the DRM shall use the UserGUID to lookup the user's userLicense. The rights in the user license are used in the playing back. The ContentKey is obtained by decrypting the embeddedContentLicense with the userKey. If no user license is found the client will fall back to try and find a normal license based on ContentGUID in the file.

If there is still no license on the machine for the file, the DRM client will hurl to the storefront to request a license in the normal manor with one exception. The difference is an additional &UserGUID=xxxx will be tacked onto the request. (This is the same method an expired subscription would have a &SubscriptionGUID=xxx tacked onto the request.) The storefront can use the userGUID to identify who purchased the content.

The store will be required to maintain a list of registered unique ClientGUIDs/MachineInfo for every licensed user. Upon receiving the userLicense request, the storefront will need to authenticate the user and determine the ClientGUID. It then checks the current ClientGUID against the registered machine list in the user's record. If the machine in question is on the list, the storefront can make a normal subscription request to the license server with Subscription_SubscriptionGUID=UserGUID and an additional parameter, Subscription_ContentKey=userKey. It should be a rare circumstance, that the user would have a known ClientGUID and not have a license... We could flag such situations as potential fraud or hack.

If the ClientGUID is not on the list but the number of current registered machines is less than the total number allowed, the storefront can add the ClientGUID to the list and increment the activation number and update the user account. It then makes a subscription license request to the license. If no more machines can be licensed to the user, the storefront shall prompts the user to de-register a machine first and then come back try again.

A storefront might choose to record a user specified Machine Name when they register a machine. They could then easily determine what machines they had registered by looking at their account status.

4 Design Choices

4.1 UserGUID.

Currently the UserName(xxx@yy.zz) and/or userID is in the user's record. A new UserGUID is needed to identify the userLicense in the local DB. The UserGUID and the UserName will also show up in the personalized content file header marking the associated user as the content owner.

4.2 Encrypted userKey.

The userKey shall be a 128-bit random AES key. It will be used to encrypt/decrypt the contentKey in the content license contained in the file Header. The userKey will be transferred to the license server encrypted with the server's public key in the same manor a regular ContentKey would be encrypted. The DRM license server will decrypt the userKey with the server's private key.

4.3 embeddedContentLicense.

The embeddedContentLicense in the content header shall have 3 parts: a CString named DRM_UserGUID with value in the 8-4-4-4-12 format, an IHXBuffer named DRM_UserName holding the login name, an IHXBuffer named DRM_UserInfo of size 173. The DRM_UserInfo buffer shall hold a base64-encoded 128-byte binary structure:

```
struct CEmbeddedContentLicense
{
  UCHAR          m_MD5Hash[16];
  int            m_Version;
  UCHAR          m_ContentKey[16];
  UCHAR          m_reserved[128-16-4-16];
};
```

The m_MD5Hash shall not be encrypted. The rest of the structure shall be encrypted with the userKey. The current version number shall be 1. The reserved space will be filled with random data.

4.4 userLicense

This userLicense is simply a subscription with a key. Subscription licenses shall have all necessary rights like the subscription to control the usage rights for all user contents. The key field shall hold the userKey, protected by the double-encryption when inserted into the license DB. When making userLicense request to the license server, the store front shall send a subscription request with Subscription_SubscriptionGUID=UserGUID and an additional parameter, Subscription_ContentKey=userKey.

4.5 Machine De-activation Protocol

A user is authorized to play userContent on up to N registered machines. We shall allow the user to change the set of machines that are registered as long as only N machines are

registered at any one time. In order to accomplish this there must be a secure way to de-register a user machine upon a user request.
The solution utilizes the existing revocation logic to accomplish this. Adding Right_RevokeLicense=1 (and not sending the content key) to the userLicnese generation should create a revoked license.

But revocation by itself is not enough. The storefront needs to receive a garneted acknowledgement after the revocation has been processed. This prevents someone from intercepting the revocation before it is inserted and "faking" de-registration of a machine.

The user initiates the process on a licensed machine with a request to de-register their machine. The storefront shall first determine the ClientGUID from the DRM info.

The store shall then check the user account's machine list for ClientGUID to ensure they have issued a license to the machine that is requesting revocation. It is an error case if they are trying to revoke a machine that has not had a license sent to it... But this error case is possible if there is a corrupted database or the database is deleted. The user would then go to "revoke" their machine but it the storefront would think it was a different machine (because of the different install ID.)

To facilitate this feature we are going to add a generalized license insertion acknowledgement method. This will allow for acknowledgment of any license insertion not just a userLicense revocation insertion.

4.6 License insertion Acknowledgment Protocol
This procedure can be used to receive notification that a license was successfully inserted into the client's machine.

The storefront will needs to pick a random number as a challenge. NOTE: it is very important that this number is random otherwise users will be able to "spoof" an insertion acknowledgement.

The challenge should be stored in a list of outstanding acknowledgments along with any data associated with the transaction (e.g. the ClientGUID that needs to be deregistered in the case of userLicense deregistration) The storefront needs to then create a license request via the license server. This is a standard license request (revocation, license, subscription) except two additional rights are added to trigger the acknowledgement protocol:

Right_InsertAckID=[storefront challenge]
Right_InsertAckURL=[url storefront is expecting an http response].

NOTE: we could just have a URL and respond to that url, leaving building the url with a parameter representing the unique random challenge to be the responsibility of the storefront.

After the DRM client has successfully processed the insertion of the license into the secure database, it will check to see if there is an InsertAckURL and ID. If they exist, the client will then hurl to the storefront with the InsertAckID appended to the end of the InsertAckURL (InsertionAcknowledgement=[storefront challenge]

The storefront can then check the pending acknowledgement list and proceed with the action that required acknowledgement of the license insertion.

5 Task Estimates

5.1 DRM Client: 5-Days

Client code to handle embeddedContentLicense, userLicense and ContentKey calculation, generate url for userLicense and embeddedContentLicense request, viewrighs change for userLicense, license insertion acknowledgement, PD userKey transfer.

5.2 License Server: 2-Days

Code to allow ContentKey for subscription license, generating embeddedContentLicense.

5.3 Packager 1-Day

Code to create the space holders in the content file header. The download manager on the client will replace the space holders with the embeddedContentLicense without writing the entire content file. The DRM_UserGUID cstring shall be zero-guid: 00000000-0000-0000-0000-00000000000. The DRM_UserInfo buffer shall be 173-byte long with all zeros. The DRM_UserName buffer shall be 256-byte filled with zeros. We will also need an ULONG32 of 1 named DynamicDRMInfo to indicate that we have some DRM_ values.

5.4) User Account Services. 2-3 Days.

Code to generate UserGUID/userKey. Code to add and manage new user account filelds UserGUID, registered ClientGUID list, numberRegistered, numberAllowed...

5.5) Store

userLicense request formation, embeddedContentLicense request formation, userLicense de-activation management.

5.6) Utility to Insert embeddedContentLicense. 2-Days.

Used by the client's download manager to locate the space holders UserGUID and UserInfo in the content file header and replace them with the real UserGUID and embeddedContentLicense.

EXHIBIT D

Subject: replay attack on Orange protocol and solutions
Date: Fri, 1 Aug 2003 15:47:52 -0400
From: Sheldon Fu <sfu@real.com>
To: 'Josh Hug' <jhug@real.com>, Alain Hamel <ahamel@real.com>, Qiang Luo <qluo@real.com>
CC: Adam Cappio <adamc@real.com>, Rahul Agarwal <rahul@real.com>

Josh pointed out that our current Orange protocol is vulnerable to replay attack --- somebody could record the 'ActivateUser' dropoff DB and later after the device is de-registered, resend the saved DB to the device to activate the device again without PC DRM knowing anything about it.

The prevention of replay attack seems not an easy task for the device:

1. Since device may not have clock/timer and we could use mass storage as communication channel, time-out check is out.
2. Same as in the PC license insertion replay prevention; we could have PC generate a unique record ID (or DB ID) each time and let device remember is a list of say last '100' IDs.
3. Alternatively, we could use a counter in DB header so that each time anybody writes to it, increases the counter and both PC and device rememebers the 'last seen counter'. This means that the dropoff DB has to be always there and can not be deleted at any time.

Big problem: Both (2) and (3) have problem when a device is reset so we will forget the list or the 'last seen counter'. It is so easy to reset a device, this is really big problem.

We could improve by (3):

4. Require that only the device could create the DB, and each time the device create a DB (after reset, or not after reset), it has to put a RANDOM number as the initial counter value. Big drawback is that it may be really tough for a device without any clock and persistence storage other than flash (like Palm) to generate a random number after reset.

Anybody has a better idea? Or we should just spec (3) and (4) in Orange and let device developers to scratch their heads to come out a random number? For a lame (and 'fun' ;-)) solution, they could ask the user to punch a button 20 times then measure the intervals between key events to come out a random number, each time a dropoff DB needs to be created.

fxd

EXHIBIT E

Subject: Helix Device DRM.ppt
Date: Fri, 15 Aug 2003 16:29:23 -0700
From: Josh Hug <jhug@real.com>
To: Ranah Edelin <ranah@listen.com>, Rahul Agarwal <rahul@real.com>, Jeff Ayars <jeffa@real.com>, Adam Cappio <adamc@real.com>, Niranjan Nagar <niranjan@listen.com>, Dave Williams <dave@listen.com>, Ryc Brownrigg <rbrownrigg@real.com>, Evan Krasts <ekrasts@listen.com>

Here is my first draft of my presentations for the Music Labels to help further the Rights discussions.

Two presentations:
1.) The Music Experience w/ Helix DRM
Explain user licensing, and present issues w/ CD burning & PD transfers from a usability perspective.

2.) Helix Device DRM
Present our Device DRM. How it works, its security, and comparison to existing standards & technologies.

TODO:
Edit, and refine.

I plan to work on it a little bit tomorrow morning, and then I will have a long flight on Sunday. Monday morning I will meet w/ Ranah for last minute adjustments, re-organization & customizations for different meetings.

Thanks for any feedback and suggestions on organization and content.

Josh

Helix™ DRM

Helix Device DRM Security Analysis

September 30, 2008

Objective

- Present Helix Device DRM Design
- Analysis of Helix Device DRM's security
- Comparison to other solutions

Design Goals

- Create familiar MP3 "like" user experience for honest users
- A user's license is transferred via the device initialization process
  - Access to this key provides access to a user's entire content library
- After the device is initialized user's content can be copied directly to the device
  - Media could be sent by the RealOne player or sent directly to the device or via alternative methods (removable media, mass storage device etc.)
  - The media format never changes
    - Same cryptographic protection and trust model

Protocol Details – Packaging

- Each file is individually encrypted with a content key (CEK)
- When a user purchases a track the track is individualized
  - Each user is assigned their own ID (UserGUID) and key (UserKey)
  - The CEK is then encrypted with the UserKey
  - This individualized blob is then inserted into the file's header (GUID, UserKey(CEK))

Protocol Details - Initialization

- Initialization of a device involves securely transferring a UserGUID and its associated UserKey
- Security
  - Use an individualized transfer key
  - Protocol protects against re-play attack
- Designed to require minimal interaction between the device and PC
  - Stateless based communication
  - Allows for easy implementation on devices with different characteristics
    - Pure mass storage devices
    - Devices with custom drivers
    - etc

Protocol Details – Playback

- The device then reads the UserGUID and UserKey(CEK) from the file header at playback time.
  - The UserKey associated with the UserGUID in the file is retrieved from the secure storage and used to decrypt the CEK
  - The CEK is then used to decrypt content before decoding and playback.

Security Analysis

- PC side of transfer protocol uses world class TR as Helix DRM
  - Includes: Individualization, obfuscation, tamper resistance, code execution in privileged mode.
- Clear compliance and robustness rules for device implementations
  - Ensured devices are secure and fit into the Helix Device DRM trust model
  - All Helix DRM devices must meet requirements in specifications

Compliance rules

- Clear compliance rules for implementations
  - More restrictive than CPRM or DTCP
    - No digital output allowed
  - Content can only be decrypted for immediate playback

Robustness rules

- Clear Robustness rules
  - Requirements are on par with SDMI & CPRM robustness requirements
  - All secrets must be protected by device hardware and software
- Software Rules & Hardware Rules robustness rules

Security level vs. Other Products

- Comparison to MS
  - User Licensing vs. Individual content licensing
  - Our user experience is better
  - Our security speaks for itself
- Comparison to Apple
  - Trust Model?

The Devices are coming

- Worked to get older versions of Helix DRM on devices
  - Were never satisfied with the user experience, thus we could not justify enough investment to bring to market.
- The usability benefits of our new version change things
  - We are actively working with multiple partners in multiple device categories to add support for our latest DRM to their devices.

Helix™ DRM

The Music Experience w/ Helix DRM

Minimizing the impact on User Experience

September 30, 2008

Objective

- Present our DRM Philosophy
- Identify the impact of different portability restrictions on usability
  - PC to PC portability model
  - CD Transfer portability
  - PD Transfer portability
- Present our recommend set of rights restrictions that will effectively protect media assets while allowing a good user experience

Philosophy

- DRM is about restricting usage
  - Our job is to protect media
  - Also enable sale of media to consumers
  - We have strived to make our restrictions seamless and invisible to users
- Replicate the usability of MP3's
  - DRM should be invisible to users who play by the rule's
  - Restrictions should only be felt when users attempt to do unauthorized actions
  - We don't want users to feel like they are in chains
    - We have learned lessons from past usability experience

Acceptable Restrictions

- Our challenge is to define restricted playback rules while still allowing for an acceptable user experience
  - We won't be successful asking users to pay for something that creates a worse user experience then what they are use to
  - To solve this, we can create software that replicates a familiar user experience
- When considering music purchase portability is the most important
  - PC to PC (our Helix DRM user licensing model)
  - PC to CD (our RealOne player Burn engine)
  - PC to PD (our Helix Device DRM specification)

The PC playback experience

- Helix DRM's is tested and approved
  - World class tamper resistance technology
  - Fully renewable via revocation
  - Has been analyzed and verified to meet bar to meet the security bar set by major movie studios and recording labels
- The Helix DRM supports three Licensing models
  - Individual Licensing
  - Subscription Licensing
  - User Licensing

Individual licensing

- Every piece of content is uniquely identifiable and can be licensed independently
- Benefits
  - Individualized control over each piece of media
- Challenges for music
  - Hard to manage large amount of similar content
    - What content has what rights? Confusing to users
  - Not a good experience for PD support
    - Have to copy a key w/ every file
    - Slow transfer rate, forced to use custom applications.
  - User experience is bad when content is moved to a second PC
    - Second PC has to be on-line
    - Must acquire each license individually
    - Hard and confusing when users want to change which computers are their authorized machines
  - Content expires individually and must be re-licensed individually

Subscription Licensing

- Individual licenses are grouped into "subscriptions"
- Access to the group of licensing is provisioned and managed by a single parent "subscription" license
- Benefits
  - Designed to allow for the dynamic update of a group of licenses on a recurring basis
  - As long as your subscription is current, you won't have to re-license your content.
- Challenges for Music
  - As in the individual licensing model Content still must be individually licensed creating a vary similar set of usability challenges
  - Solves the problem of licensing content on a recurring basis, but does not go far enough to improve portability and rights management

User Licensing

- Files are no-longer individually licensed and controlled
  - Same benefits as Subscription licensing, plus much simpler portability and rights management.
  - User license is protected with the same security and rights enforcement as all Helix DRM licenses
- Usability and portability benefits
  - One User License provides access to an entire body of content with similar rights
  - Simple for users to understand
    - Especially for music when users have hundreds of tracks
  - Effective protection and restrictions
    - Can express full set of rights restrictions on single User License – applied to all the content
  - Easy to register or de-register a machine
- Challenges
  - No control over individual tracks

User Licensing - PC User Experience

- User licensed content is "personalized" or "bound" to the Licensed User at the time of download.
  - The Content contains information about the User to whom the content is licensed, and it also contains the encrypted Content Key required to decrypt the content during playback.
  - The Content Provider assigns a unique User Key to each Licensed User.
  - The User Key is used to encrypt the Content Key in every User Licensed content file.
- To consume the User Licensed content file, the User must register their PC with the Content Provider.
  - Upon registration, the User will receive a License with the User Key and associated Rights.
  - The User License is kept in Secure Storage on the PC and is bound to the PC and to the User.
  - Access to the User License enables access to User's entire User Licensed content catalog.

User Licensing – PC to PC

- If a User wishes to use their content catalog on a different PC, they need to do two things:
  - Copy their User Licensed content to the new machine.
  - Register the new PC with the Content Provider, thus obtaining a License with the User Key.
- The user is restricted to register "N" machines.
  - If the User has already registered "N" machines and attempts to register their "N+1" machine, they will be asked to un-register one of their already registered machines before registering a new one.
  - An un-registered machine removes the User's License disabling playback of User's content until the User chooses to re-license their machine.

Music Portability

- Only allowing music playback on PCs limits usability
- Two portability methods are needed for digitally purchased music
  - Transfers to CD
    - CD players are everywhere and cheap
    - People currently expect to be able to play music they buy in CD players
  - Transfers to PD
    - PD's are becoming more popular with Users
    - Need to provide a secure method of transferring purchased music to these Devices.

Transfers to CD

- The Problem:
  - The restrictions we place on burning should not deter people from purchasing music
- The following usage restrictions have been proposed and we have investigated their impact on the user experience
  - Burn restrictions on individual tracks (i.e.)
    - 10 burns per track on one PC
    - 10 burns per track
    - 10 burns per track on each registered machine
  - Burn restrictions on play lists
    - 10 burns per play list
  - Combinations of the above

Usability Analysis – Burn Limits

- Generally user licensing was not designed to track individual counts associated with individual pieces of content
  - Requirement would eradicate many of the usability benefits associated with user licensing
  - Every piece of content would need an additional license to allow and count burning for each track
  - Significant amount of complexity is added when the track limits have been exceeded
- N burns on one PC
  - Clearly limits users ability to burn wherever they want
  - User must anticipate where they will do the majority of their burning – long term customer service problem:
    - What do we do when a user buys a new PC with better hardware?

Usability Analysis – Burn Limits

- N total burns for each track
  - Requires user to be connected during burn
  - Slows down burn process
  - Requires Users to authenticate with service whenever they burn
  - Requires an on line system to track the number of burns users have left on a track by track basis
  - A much more complex and slower user experience when compared with burning unprotected file
- N burns for each play list
  - No noticeable effect to the burn experience unless a play list is recorded N times
  - Effectively prevents people from pressing 100s of cd's for their friends.

Usability Analysis – Burn Limits

- General Issues with Burn Limits
  - Legitimate users should not feel limits
    - They are designed to be felt by people that break the rules
    - What is the difference between setting a limit of 10 and unlimited?
  - Confusing to customer since they just want to own the content and use it legitimately
  - Customer feels limited and confined, rather than free to use what they purchased

Transfers to PD

- The Problem:
  - Secure files can only be transferred to secure Digital Music Players. What are appropriate restrictions to be placed on track that a user has legitimately purchased.
- Helix DRM uses a registration model
  - Secure method is defined to allow a PC to securely provision a PD with a User Key.
  - Once the PD has been provisioned with a User Key, i.e. "registered" to the User, the device gains access to a User's entire library of content.
    - The content is simply directly transferred from the PC to the PD using any means available.
  - The process of de-registering a device is simply the inverse of registration.
    - The PC needs to verify that the PD is no longer capable of playing the User's content.

Familiar User Experience

- We want to provide familiar user experience to honest users
  - It should be silent and require no extra user interaction
- Restricting transfers sounds simple… But it quickly can grow into a very complicated user experience to express.
  - We are most concerned with keeping the user experience simple for legitimate users.
  - Multiple users can be registered on the same PC…
    - Therefore we believe we have to treat PD's as an extension of the PC's rights rather than an individual user's.
  - We have tried to diagram other methods and they become WAY to complicated.
    - Definitely not silent methods

Usability Analysis – Transfer Limits

- There are three restrictions that seem possible when we implement restrictions while considering the PD to be an extension of the PC's user set
  - Max Number of PD for each registered PC
  - When transferring a new user to a PD, the transferring PC must remove all users from the PD that are not currently registered on the transferring PC.
    - I.e. the PC must sync the users on the PD w/ the PC's set of users when it changes users on a PD.
  - Allow direct transfer to all secure and approved devices

Restriction 1 – Max Number of PD's

- We could envision a restriction on the number of devices a user could connect to a PC
  - Each PC would have a limited number of devices it could communicate with
  - When the users used up all their devices, they would have to un-register a device with the computer before they could register another.
- Challenges
  - While at first this sounds reasonable, it is not without issues.
    - Main issue is it requires user interaction for device registration

Restriction 1 – Max Number of PD's

- Challenges (cont)
  - We don't see most users having that many PD's...
    - We can't just register the PD's silently though...
    - They need to know that they are using one of the PC's limited number of slots.
  - On a multi-user PC, one user might use up all the allowed devices
    - Other users are left in the lurch.
  - No method for restoring device count for new PD's after one has been lost, broken or otherwise disposed of.
    - We don't have a secure method to allow our customer service to re-set their PC's device count.
    - Unless we created a back door in the security.

Restriction 2 – Sync Users

- Registering a user on a PD means synchronizing the PD's users w/ PC you are transferring from
  - This implies unlimited number of transfers but restricts devices ability to aggregate content from a number of different users.
  - If an old user is not on the new PC being transferred from, transferring a new user from the new PC will erase the old user from the device.
- Challenges
  - In some cases content would stop working on the device after syncing with a new PC.

Restriction 2 – Sync Users (cont)

- Users would run into this restriction in the following situation:
  - PC1 - UA & UB registered; PC2 - UB & UC registered
  - Register device to PC1... UA & UB are transferred
  - Go to PC2
  - UB content would transfer fine with no problems from PC2.
    - But when UC content is queued for transfer, we detect that the device will loose the rights to UA content if they proceed.
    - The user is then informed that activating UC content will mean UA content will no longer work on their device. (to remedy in help, they can either authorize UA on the new PC, or authorize UC on their old PC.)

Restriction 3 – Only secure devices

- Enabling transfer to all secure devices creates the best user experience
  - Fulfils the silent of silent registration
  - Users do not have that many devices
  - The content is still protected
  - We can track its usage when they try to play it on PC's
  - Devices have to connect to a registered PC to get a user's credentials

Recommendation

- User experience is the most important
  - The best user experience is achieved with
    - Max of 3 registered PC's
    - Unlimited burning
    - Unlimited transfers to Secure PD's

EXHIBIT F

Subject: UserGUID byte order fix CR
Date: Tue, 30 Sep 2003 15:45:07 -0700
From: Qiang Luo <qluo@real.com>
To: Sheldon Fu <sfu@real.com>, Alain Hamel <ahamel@real.com>
CC: Josh Hug <jhug@real.com>, Adam Cappio <adamc@real.com>

Please review the fix. Thanks, Qiang.

Diff.txt:

```
Index: trsrc/enforce.cpp
===================================================================
RCS file: /home/dormroom/basicdrm/trsrc/enforce.cpp,v
retrieving revision 1.179
diff -u -r1.179 enforce.cpp
--- trsrc/enforce.cpp 30 Sep 2003 12:00:06 -0000 1.179
+++ trsrc/enforce.cpp 30 Sep 2003 22:43:01 -0000
@@ -4530,6 +4530,16 @@
     TR_DELETE(pTacRecord);
 }

+void Tr_PackGUID(GUID* pGUID)
+{
+    // I don't have DwToNet and uuid stuff in tr so I'm just doing to
do bswap for LITTLE_ENDIAN
+#ifdef LITTLE_ENDIAN
+    bswap4(pGUID->Data1);
+    bswap2(pGUID->Data2);
+    bswap2(pGUID->Data3);
+#endif
+}
+
 void Tr_SetupTransferBuffer(void** params, int* theErr, UCHAR*
pMachineInfo)
 {
     char pszDevicePublicKeyName[19] =
{'D','e','v','i','c','e','R','S','A','P','u','b','l','i','c','K','e','
y',0};
@@ -5124,6 +5134,7 @@

         if (pUserGUIDBuffer && totalUsersOnPD > 0 && *UserCount >=
totalUsersOnPD)
         {
```

```
+           // Do not need to pack the GUID for output. It is already
in net byte order.  We just dump them to the caller
            Tr_memcpy(pUserGUIDBuffer, pUsersOnPD,
totalUsersOnPD*sizeof(GUID));
          }
          else if(totalUsersOnPD > 0 && (!pUserGUIDBuffer || *UserCount
< totalUsersOnPD))
@@ -5155,6 +5166,8 @@
            // coyp out the user GUIDs
            for (i=0; i<totalUsersOnPC; i++)
            {
+               // pack the GUID for output
+               Tr_PackGUID((GUID*)(pUsersOnPC+i*2));
                Tr_memcpy(pUserGUIDBuffer+i, pUsersOnPC+i*2,
sizeof(GUID));
            }
          }
@@ -5173,6 +5186,18 @@
          }
          else
          {
+           // coyp out the user GUIDs
+           if (!*theErr)
+           {
+               for (i=0; i<totalUsersOnPC; i++)
+               {
+                Tr_DecrementRight((char*)(pUserGUIDBuffer+i), NULL,
pszTransferToPD, sizeof(pszTransferToPD), theErr, pMachineInfo);
+                // pack the GUID for device messages and output
+                Tr_PackGUID((GUID*)(pUsersOnPC+i*2));
+                Tr_memcpy(pUserGUIDBuffer+i, pUsersOnPC+i*2,
sizeof(GUID));
+               }
+           }
+
            TR_MALLOC(&pSessionKey, 16);
```

```
            GenerateRandomBuffer(pSessionKey, 16, NULL, pMachineInfo,
theErr);
            // form messages out
@@ -5193,15 +5218,6 @@
                OrangePackMessages(pMessageOut, *pMsgOutSize, theErr);
            }
 #endif
-           // coyp out the user GUIDs
-           if (!*theErr)
-           {
-               for (i=0; i<totalUsersOnPC; i++)
-               {
-                 Tr_memcpy(pUserGUIDBuffer+i, pUsersOnPC+i*2,
sizeof(GUID));
-                 Tr_DecrementRight((char*)(pUserGUIDBuffer+i), NULL,
pszTransferToPD, sizeof(pszTransferToPD), theErr, pMachineInfo);
-               }
-           }
          }
          *UserCount = totalUsersOnPC;
      }
```

EXHIBIT G

Subject: RE: [Fwd: Machine Reg/Dereg Issues]
Date: Wed, 08 Oct 2003 14:30:35 -0700
From: Qiang Luo <qluo@real.com>
To: Alain Hamel <ahamel@real.com>, Josh Hug <jhug@real.com>
CC: Adam Cappio <adamc@real.com>
References: <5.1.0.14.2.20031008103838.02fdeb60@mailone.real.com>

Al,

Please review the fix.

Adam uses embedded player to play the license file and everything works fine.

Listen uses jscript and ierpplug to load the core to call RMADRMLicenseManager::AddLicense directly. There is no player made available from the pContext. nPlayerCount = pEngine->GetPlayerCount() returns 0.

I tested using both IE and RealOne against both Adam's and Listen's storefront. All works. However, there is one problem: when I use RealOne player to de-register machine from Listen, it hurl to their page in a new window. It works correctly if I use IE. I have set the target frame to "_self". What else I can do to force the page in the same window?

Thanks,

Qiang

At 11:23 AM 10/8/2003 -0700, Alain Hamel wrote:
>Qiang,
>
>so nPlayerCount = 0 after executing the line:
>
> > nPlayerCount = pEngine->GetPlayerCount();
>
>Wow.
>
>That would mean that a player has never been created. Interesting. I just
>looked through the embedgui code, and then the rpplmgr and
>then the rprmapl code and it has changed quite a bit since I last looked
>at it. I think it's possible that if there is no url within
>the embed tag (object tag) then no player is created.
>
>And if no player is created go nuts. Call CreatePlayer. And wow are you
>lucky calling createplayer calls _initalize so you should be
>good to go.
>
>-----Original Message-----
>From: Qiang Luo [mailto:qluo@real.com]
>Sent: Wednesday, October 08, 2003 9:58 AM
>To: Josh Hug; Alain Hamel
>Cc: Adam Cappio
>Subject: Re: [Fwd: Machine Reg/Dereg Issues]
>

>
>Josh & Al,
>
>The insertion ack works against Adam's store. The pPlayer is from (see the
>attached code)
>m_pContext->QueryInterface(IID_IRMAPlayer, (void**)&pPlayer);
>
>However, when using listen's store, get player from all three
>methods(passed to us, from pContext, from pEngian) fails.
>
>One fix that I can think of is when QI player attempts all fail, we will
>create a player from pEngine and then hyper navigate via the new
>player. Any suggestions?
>
>Thanks,
>
>Qiang
>
>
> >Date: Wed, 08 Oct 2003 10:37:22 -0700
> >To: Matt Faso <mfaso@real.com>
> >From: Qiang Luo <qluo@real.com>
> >Subject: Re: [Fwd: Machine Reg/Dereg Issues]
> >Cc: Josh Hug <jhug@real.com>, Brad Pitzel <bpitzel@real.com>,
> >jchasen@real.com, Anuj Khandelwal <akhandelwal@real.com>
> >
> >Matt,
> >
> >There is indeed a DRM bug. DRM attempts to send the ack but failed to get
> >a HXPlayer to kick off the hurl. I will file a bug and fix it. Could you
> >try to use the silent option for now to see whether the silent method
> >works? The syntax, before urlbase64, is
> >Silent:http://172.23.104.195/mstore/deregister.php?deregister=...."
> >
> >Thanks,
> >
> >Qiang

Diff.txt:

```
Index: revcli.cpp
===================================================================
RCS file: /home/dormroom/basicdrm/revcli.cpp,v
retrieving revision 1.5
diff -u -r1.5 revcli.cpp
--- revcli.cpp   21 Aug 2003 13:04:29 -0000 1.5
+++ revcli.cpp   8 Oct 2003 21:22:35 -0000
@@ -95,24 +95,31 @@
      ret = m_LastError;
      }

+     // QI hypernavigate directly
+     IRMAHyperNavigate2* pHyper2 = NULL;
+     if (SUCCEEDED(ret) && !m_bSilent)
+     {
+     m_pContext->QueryInterface(IID_IRMAHyperNavigate2,
(void**)&pHyper2);
+     }

      // player for hypernavigate
      IRMAPlayer* pPlayer = NULL;
+     IRMAPlayer* pNewPlayer = NULL;

-     if (SUCCEEDED(ret) && !m_bSilent && pRMPlayer)
+     if (SUCCEEDED(ret) && !m_bSilent && !pHyper2 && pRMPlayer)
      {
      // passed to us
      pPlayer = pRMPlayer;
      pPlayer->AddRef();
      }

-     if (SUCCEEDED(ret) && !m_bSilent && !pPlayer)
```

```
+     if (SUCCEEDED(ret) && !m_bSilent && !pHyper2 && !pPlayer)
      {
      // QI from pContext
      m_pContext->QueryInterface(IID_IRMAPlayer, (void**)&pPlayer);
      }

-     if (SUCCEEDED(ret) && !m_bSilent && !pPlayer)
+     if (SUCCEEDED(ret) && !m_bSilent && !pHyper2 && !pPlayer)
      {
      // player from pEngine
      BOOL bFound = FALSE;
@@ -160,15 +167,24 @@
      }
      }

-     // hypernavigate
-     if (SUCCEEDED(ret) && !m_bSilent && pPlayer)
+     if (SUCCEEDED(ret) && !m_bSilent && !pHyper2 && !pPlayer &&
pEngine)
+     {
+     // we are out of luck! we have to create our player from
pEngine...
+     pEngine->CreatePlayer(pNewPlayer);
+     pPlayer = pNewPlayer;
+     }
+
+     if (SUCCEEDED(ret) && !m_bSilent && !pHyper2 && pPlayer)
+     {
+     pPlayer->QueryInterface(IID_IRMAHyperNavigate2,
(void**)&pHyper2);
+     }
+
+     if (SUCCEEDED(ret) && !m_bSilent && pHyper2)
      {
-     IRMAHyperNavigate2* pHyper2 = NULL;
      IRMACommonClassFactory* pCCF = NULL;
```

```
      IRMAValues* pRequest = NULL;


-     pPlayer->QueryInterface(IID_IRMAHyperNavigate2,
(void**)&pHyper2);
-     pPlayer->QueryInterface(IID_IRMACommonClassFactory,
(void**)&pCCF);
+     m_pContext->QueryInterface(IID_IRMACommonClassFactory,
(void**)&pCCF);


      if (pCCF && pHyper2)
      {
@@ -185,13 +201,18 @@
           pRequest);
      }


-     PN_RELEASE(pHyper2);
      PN_RELEASE(pRequest);
      PN_RELEASE(pCCF);
     }


+     if (pNewPlayer)
+     {
+     pEngine->ClosePlayer(pNewPlayer);
+     }
+
     PN_RELEASE(pEngine);
     PN_RELEASE(pPlayer);
+     PN_RELEASE(pHyper2);


     return ret;
 }
```

code.txt:

```
PN_RESULT
RevokeClient::Initialize(const char* pRevokeURL, IUnknown* pContext,
RevokeClientResponse* pResponse, IRMAPlayer* pRMPlayer, BOOL bSilent)
{
    const char pszSilent[8] = {'S','i','l','e','n','t',':',0};
    const char pszSelf[6] = {'_','s','e','l','f',0};
    const char* pURL  = NULL;
    PN_RESULT ret     = PNR_OK;

    m_pContext = pContext;
    m_pContext->AddRef();

    // set mode
    m_bSilent = bSilent;
    pURL = pRevokeURL;
    if (strstr(pRevokeURL, pszSilent))
    {
     pURL += 7;
     m_bSilent = TRUE;
    }

    m_pRevokeURL    = ::new_string(pURL);
    m_pResponse     = pResponse;

    // pEngine
    IRMAClientEngine* pEngine = NULL;
    ret = m_pContext->QueryInterface(IID_IRMAClientEngine, (void**)
&pEngine);

    // silent http request
    if (SUCCEEDED(ret) && m_bSilent)
    {
```

```
      pEngine->CreatePlayer(m_pNewPlayer);

         if (pRMPlayer)
         {
            IUnknown* pCtxt = NULL;
            pRMPlayer->GetClientContext(pCtxt);
            m_pNewPlayer->SetClientContext(pCtxt);
            PN_RELEASE(pCtxt);
         }

      m_pNewPlayer->AddAdviseSink((IRMAClientAdviseSink*) this);

      IRMAErrorSinkControl* pSinkCtl = NULL;
      m_pNewPlayer->QueryInterface(IID_IRMAErrorSinkControl, (void**)
&pSinkCtl);
      pSinkCtl->AddErrorSink((IRMAErrorSink*) this, PNLOG_EMERG,
PNLOG_ERR);
      pSinkCtl->Release();

      m_LastError = m_pNewPlayer->OpenURL(m_pRevokeURL);

      if (m_LastError == PNR_OK)
      {
          m_LastError = m_pNewPlayer->Begin();
      }

      ret = m_LastError;
    }

    // QI hypernavigate directly
    IRMAHyperNavigate2* pHyper2 = NULL;
    if (SUCCEEDED(ret) && !m_bSilent)
    {
      m_pContext->QueryInterface(IID_IRMAHyperNavigate2,
(void**)&pHyper2);
    }
```

```
// player for hypernavigate
IRMAPlayer* pPlayer = NULL;
IRMAPlayer* pNewPlayer = NULL;

if (SUCCEEDED(ret) && !m_bSilent && !pHyper2 && pRMPlayer)
{
 // passed to us
 pPlayer = pRMPlayer;
 pPlayer->AddRef();
}

if (SUCCEEDED(ret) && !m_bSilent && !pHyper2 && !pPlayer)
{
 // QI from pContext
 m_pContext->QueryInterface(IID_IRMAPlayer, (void**)&pPlayer);
}

if (SUCCEEDED(ret) && !m_bSilent && !pHyper2 && !pPlayer)
{
 // player from pEngine
 BOOL bFound = FALSE;
 UINT16 nPlayerCount = 0;
 UINT16 sourcecount = 0;
 UINT16 streamcount = 0;

 nPlayerCount = pEngine->GetPlayerCount();

 for (int i = 0; i < nPlayerCount && !bFound; i++)
 {
     IUnknown* pUnk = NULL;
     pEngine->GetPlayer(i, pUnk);
     if (pUnk)
     {
       PN_RELEASE(pPlayer);
```

```
            pUnk->QueryInterface(IID_IRMAPlayer, (void**)&pPlayer);
            if (pPlayer)
            {
                sourcecount = pPlayer->GetSourceCount();
                for (int j = 0; j < sourcecount && !bFound; j++)
                {
                  IRMAStreamSource* pSource = NULL;
                  IUnknown* pUnk2 = NULL;
                  pPlayer->GetSource(j, pUnk2);
                  if (pUnk2)
                  {
                      pUnk2->QueryInterface(IID_IRMAStreamSource,
(void**)&pSource);
                      if (pSource)
                      {
                        streamcount = pSource->GetStreamCount();
                        if (streamcount)
                        {
                            // active player
                            bFound = TRUE;
                        }
                      }
                  }
                  PN_RELEASE(pSource);
                  PN_RELEASE(pUnk2);
                }
            }
          }
          PN_RELEASE(pUnk);
      }
    }


    if (SUCCEEDED(ret) && !m_bSilent && !pHyper2 && !pPlayer &&
pEngine)
    {
```

```
      // we are out of luck! we have to create our player from
pEngine...
      pEngine->CreatePlayer(pNewPlayer);
      pPlayer = pNewPlayer;
    }

    if (SUCCEEDED(ret) && !m_bSilent && !pHyper2 && pPlayer)
    {
      pPlayer->QueryInterface(IID_IRMAHyperNavigate2,
(void**)&pHyper2);
    }

    if (SUCCEEDED(ret) && !m_bSilent && pHyper2)
    {
      IRMACommonClassFactory* pCCF = NULL;
      IRMAValues* pRequest = NULL;

      m_pContext->QueryInterface(IID_IRMACommonClassFactory,
(void**)&pCCF);

      if (pCCF && pHyper2)
      {
          pCCF->CreateInstance(IID_IRMAValues, (void**)&pRequest);
      }

      if (pHyper2 && m_pRevokeURL && pRequest)
      {
          pRequest->SetPropertyULONG32("IsLicenseAcknowledgement", 1);
          ret = pHyper2->Execute((const char*)m_pRevokeURL,
            pszSelf,
            NULL,
            NULL,
            pRequest);
      }

      PN_RELEASE(pRequest);
```

```
        PN_RELEASE(pCCF);
    }

    if (pNewPlayer)
    {
        pEngine->ClosePlayer(pNewPlayer);
    }

    PN_RELEASE(pEngine);
    PN_RELEASE(pPlayer);
    PN_RELEASE(pHyper2);

    return ret;
}
```